MethodSCRIPT Examples - C#

PalmSens
Compact Electrochemical Interfaces

Last revision: March 22 2019

www.palmsens.com

1.1 Contents:

The examples in the "/MethodSCRIPTExamples-DotNet" folder demonstrate basic communication with the EmStat Pico like how to connect to the device, send method scripts to the device, run measurements on the device, read the response from the device, using simple plot objects to plot the data and close a connection to the device.

1.2 Examples:

1.2.1 Example 1: Basic Console Example (MSConsoleExample)

This example demonstrates how to implement USB serial communication to

- Establish a connection with the device
- Write the input measurement parameters to the device
- Read and parse the measurement response from the device

This does not include error handling, method validation etc.

1.2.2 Example 2: Plot Example (MSPlotExample)

In addition to the basic communications as in the above example, this plot example demonstrates how to implement the plot object (using the OxyPlot library for windows forms).

1.2.3 Example 3: EIS Console Example (MSEISExample)

This console example demonstrates sending, receiving and parsing data for a simple EIS measurement.

1.2.4 Example 4: EIS Plot Example (MSEISPlotExample)

This example demonstrates the implementation of OxyPlot to show the EIS measurement response on Nyquist and Bode plots.

1.3 Communications

The example projects are built using Visual Studio 17.

1.3.1 Connecting to the device

The examples use the System.IO.Ports library for serial communication with the device. The read time out for the port is set to 7000ms. In case of measurements with long response times, the read time out can be set higher.

To prevent your program from crashing it is recommended to use a try catch sequence when connecting to a port, this way the port will be closed again when an exception occurs.

1.3.2 Sending a script file

The input parameters for a measurement can be read from a script file stored in the path "ScriptFilePath" and then sent to the device. In this example the script file is stored in the same location as the location of the application executable.

1.3.3 Receiving response

Once the script file is sent to the device, the measurement response can be read continuously from the device.

```
readChar = SerialPortEsP.ReadChar();
```

1.3.4 Parsing the response

Each line of response returned by the method `ReadResponseLine()` is parsed further if it is identified to be a data package.

Here's a sample response (raw data) from a Linear sweep voltammetric measurement.

eM0000\n
Pda7F85F3Fu;ba48D503Dp,10,288\n
Pda7F9234Bu;ba4E2C324p,10,288\n
Pda806EC24u;baAE16C6Dp,10,288\n
Pda807B031u;baB360495p,10,288\n
*\n
\n

While parsing the response, various identifiers are used to identify the type of response packages. For example, In the above sample response package,

1. 'e' marks the beginning of a response.
2. 'M' marks the beginning of a measurement loop.
3. 'P' marks the beginning of a row of data package.
4. "*\n" marks the end of measurement.
5. "\n" marks the end of response.

The following information can be found in the data packages received from the device.

- Potential (set cell potential in V)
- Current (measured current in A)

In case of Impedance spectroscopy measurements, the following data values can be obtained from the response.

- Frequency (set frequency in Hz)
- Real part of complex Impedance (measured impedance Ohm)
- Imaginary part of complex Impedance (measured impedance in Ohm)

The following meta data values if present can also be obtained from the data packages.

- CurrentStatus (OK, underload, overload, overload warning)
- CurrentRange (the current range in use at the moment)
- Noise (Noise)

1.3.4.1 Parsing the parameter values

Each row of data package begins with the header 'P'. The parameters from the data package line can be then split in to tokens based on the delimiter ';'. Each of the parameters separated from the package line can be then parsed to get the actual values of the parameters.

The initial two characters of every parameter identifies the parameter.

```
paramIdentifier = parameter.Substring(startingIndex, 2);
```

The potential readings are identified by the string "da"
The current readings are identified by the string "ba"
The frequency readings are identified by the string "dc"
The real impedance readings are identified by the string "cc"
The imaginary impedance readings are identified by the string "cd"

For example, in the sample package seen above, the parameter identifiers are
'da7F85F3Fu' - 'da' Potential reading and
'ba48D503Dp,10,288' – 'ba' current reading.

The following 8 characters hold the parameter value

The parameter value for current reading (8 characters) from the above sample package is '48D503Dp'.

This value is further parsed to retrieve the actual parameter value with the respective unit prefix.

```
double paramValueWithPrefix = ParseParamValues(paramValue);
```

The SI unit prefix from the package can be obtained from the parameter value at position 8

```
    char strUnitPrefix = paramValueString[7];
```

In the above sample package, the unit prefix for current data is 'p' which is 1e-12 A.

After obtaining the parameter identifier and the parameter value from the package, the meta data values if present can be parsed. Meta data values if present are separated by the demiliter character ','

```
ParseMetaDataValues(parameter.Substring(10));
```

1.3.4.2 Parsing the meta data values

The meta data values are separated based on the delimiter ',' and each of the values is further parsed to get the actual value. The first character of each meta data value identifies the type of meta data.

'1' – status
'2' – Current range index
'4' - Noise

The status is 1 character hex bit mask. It is converted to long int.

The status can be obtained as shown in the code below.

```
string status = "";
long statusBits = (Convert.ToInt32(metaDatastatus[1].ToString(), 16));
```

0 indicates OK

```
if ((statusBits) == (long) ReadingStatus.OK)
    status = "OK";
```

1 indicates overload

```
if ((statusBits & 0x2) == (long) ReadingStatus.Overload)
    status = "Overload";
```

4 indicates underload

```
if ((statusBits & 0x4) == (long) ReadingStatus.Underload)
    status = "Underload";
```

and 8 indicates overload warning (80% of maximum).

```
if ((statusBits & 0x8) == (long) ReadingStatus.Overload_Warning)
    status = "Overload warning";
```

For example, in the above sample, the available meta data values for current data are, 10,288. The first meta data value is 10.

1 – meta data status – 0 indicates OK.

The meta data type current range is 2 characters long hex value. If the first bit high (0x80), it indicates a high speed mode current range.

For example, in the above sample, the second meta data available is 288.

2 – indicates the type – current range
88 – indicates the hex value for current range index – 1mA. The first bit 8 implies that it is high speed mode current range.

Sample outputs:

The following are some sample data packages and their correponding outputs.

(Please note that the spaces in the data packages are added for readability).
A row of data package from LSV measurement response

P da7F85F3Fu ; ba4BA99F0p , 10 , 288

Output:
E (V) = -4.999E-01
i (A) = -4.999E-01
Status : OK
CR : 1mA (High speed)

A row of data package fom EIS measurement response

P dc8030D40 ; cc8088C7Cm , 12 , 289 ; cdA9279D1u ,12 , 289

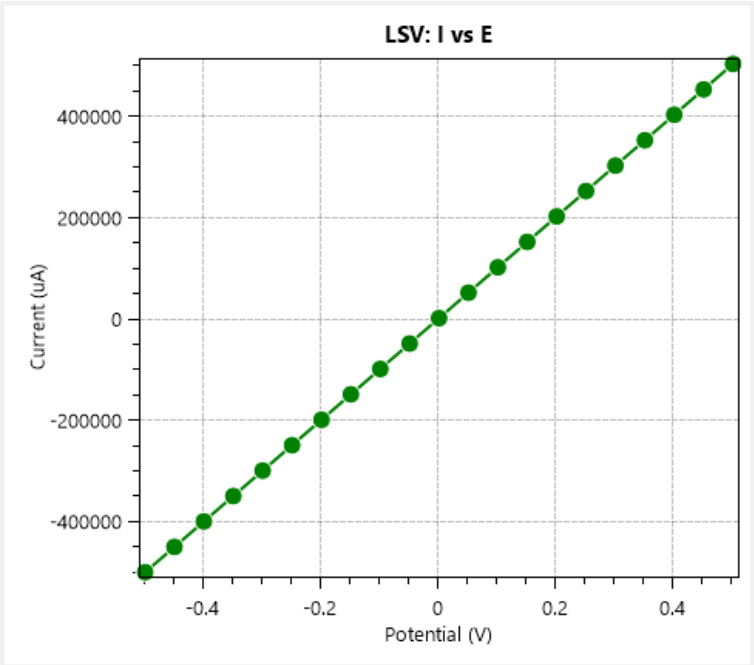
Output:
Frequency (Hz) : 200000.00
Z' (Ohm) : 2.000E+05 Status : Overload CR : 5mA
Z'' (Ohm) : 2.000E+05 Status : Overload CR : 5mA

1.3.5 Plotting the response

The OxyPlot library from NuGet packages is used in this example for showing a simple plot of the measurement response parameters.

```
using OxyPlot;
using OxyPlot.Axes;
using OxyPlot.Series;
```

Here's a sample plot with the response for a Linear sweep measurement on EmStat Pico.



In case of an Impedance spectroscopy measurement, two separate plot objects and plot models are created. One for Nyquist plot and the other for Bode plot.

The plot type is set to 'Cartesian' type in case of Nyquist plot model and 'XY' type in case of Bode plot model.

```
plotModel.PlotType = plotModel.Equals(NyquistPlotModel) ?
PlotType.Cartesian : PlotType.XY;
```

Nyquist plot axes are set to linear axes and Bode plot axes are set to logarithmic axes.

Here's are the Nyquist and Bode plots with the response for an Impedance sprctroscopy measurement on EmStat Pico.